Inheritance
Lecture 31

Borrowing from slides by Alan Hu, Kurt Eiselt, Paul Carter, and Tamara Munzner

News
- Homework 3 is online
  - Due December 4
- Final exam: Dec 11, 8:30 am, OSBO A
  - Length: 2:30

Reading Assignments
- Reading for this week: interfaces
  - Edition 2: Ch 11.1-11.3
- Reading for this/next week: inheritance
  - Edition 3: Ch 10
  - Edition 2: Ch 13

Recap: Interfaces in Real Life
- What does it mean
  - When a product says it's "USB compatible"?
  - When a gas station sells "regular unleaded (87 octane) gas"?
  - When you buy a CD that says "Compact Disc Digital Audio"?
  - The producer promises that the product has certain features and behaviors.
  - If the user uses only those features and behaviors, then everything should work right.

Recap: Interfaces as Contracts
- If you buy a "USB" product, but the plug is shaped wrong, who is responsible?
- If you put 87 octane gas in a car that runs on diesel, who is responsible?
- If you buy a CD, but it actually installs secret spyware on your computer, who is responsible?
- Producer promises to supply certain features
- Consumer promises to use only those features.

Recap: Interface “Feedable”
- Let’s create a new interface type, called Feedable.
- All Feedable classes must have the following two methods:
  - public String getFavoriteFood()
  - public void feedFood(String food)
- In the interface, leave off the “public” since they’re always public.
- Only public properties are part of an interface!
Recap: Multiple Interfaces
- A physical object can have multiple interfaces
  E.g., a digital camera with a USB port, a Firewire port, an S-video output, a CompactFlash slot, etc.
- A Java object can implement multiple interfaces
- In Java: second interface Pettable

Recap: instanceof
- Can we test which of our pets can be patted?
  - Yes: Java allows you to query which class or interface an object belongs to!
  - You can check which class an object belongs to using the `instanceof` operator:

```java
if (pet instanceof Pettable) {
    Pettable pPet = (Pettable) pet;
    pPet.pet();
}
```

Recap: Multiple Interfaces, Casting

<table>
<thead>
<tr>
<th>pets</th>
<th>pets</th>
<th>pets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dog</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pPet</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cat</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Legend:
- Feedable (red)
- Pettable (blue)

Objectives
- Understand the basic concept of inheritance, and how to `extend` classes.

New Concept: Inheritance
- Inheritance in real life: stuff you get from your parents, e.g.:
  - Harry Potter inherited his father’s hair and his mother’s eyes.
  - He inherited his mother’s sense of humor.
- (Think of inheriting characteristics and behaviors, not money and real estate.)

Inheritance in Java
- You can declare a new class as an extension of an existing one.
- The new class automatically inherits all the instance fields and methods of the old class.
- The new class can add/change fields and methods.
```java
public class ChildClass extends ParentClass {
    ...
    put any additional fields and methods here
    ...
}
```
Inheritance Terminology

- The child class extends the parent class.
- The parent class is called the superclass.
- The child class is called the subclass.
- The child class (subclass) inherits stuff from its parent (superclass), so it has more stuff (fields and methods).
- Mnemonic: your parents are your superiors.

Inheritance Example

- Suppose you are writing a simulation/game.
- There will be various things on the playing field, so you’ll have a Thing class, with instance fields like its position and direction.

Inheritance Example

- Suppose you are writing a simulation/game.
- There will be various things on the playing field, so you’ll have a Thing class, with instance fields like its position and direction.
- Some of the things will move, so you might write a MovingThing class, which is similar, but also has attributes like velocity.
- You might make an AcceleratingThing class, to make it easy to model, e.g., a spaceship.

Accessing the Superclass

- Even though the subclass has all the instance fields and methods of the superclass, Java still thinks of it as a separate class. So, the subclass can’t directly access anything private!
- What do you do?
  - Use the public accessor/mutator methods.
  - Have the superclass make things public.
  - (Have the superclass make things protected.)

Special Case: Superclass Constructor

- Recall the special use of this in constructors?
  - this as first line of constructor calls a different constructor for the same object
  - E.g., UBCStudent class
Special Case: Superclass Constructor
- Recall the special use of `this` in constructors?
  - `this` as first line of constructor calls a different constructor for the same object
  - E.g., UBCStudent class
- Similar trick to call a superclass constructor using the `super` keyword:
  - `super` call must be first line of constructor

Inheritance Hierarchy in UML
```
Thing
    ▲
  MovingThing
      ▲
  AcceleratingThing
```
“is-a” relation
(This is where the terms subclass and superclass really come from.)